```
module fifo_sync #(
    parameter DATA_WIDTH = 8,     // Width of each data word
    parameter FIFO_DEPTH = 16     // Depth (number of words)
)(
    input wire clk,
    input wire rst,

    input wire wr_en,             // Write enable
    input wire rd_en,             // Read enable

    input wire [DATA_WIDTH-1:0] din,   // Data input
    output reg [DATA_WIDTH-1:0] dout,  // Data output

    output wire full,
    output wire empty,
    output wire almost_full,
    output wire almost_empty
);

    // Memory array
    reg [DATA_WIDTH-1:0] fifo_mem [0:FIFO_DEPTH-1];

    // Write and read pointers (need only log2(FIFO_DEPTH) bits)
    localparam PTR_WIDTH = $clog2(FIFO_DEPTH);
    reg [PTR_WIDTH-1:0] wr_ptr;
    reg [PTR_WIDTH-1:0] rd_ptr;

    // Counter to track number of elements in FIFO
    reg [PTR_WIDTH:0] fifo_count; // Can count up to FIFO_DEPTH

    // Write operation
    always @(posedge clk) begin
        if (rst) begin
            wr_ptr <= 0;
        end else if (wr_en && !full) begin
            fifo_mem[wr_ptr] <= din;
            wr_ptr <= wr_ptr + 1;
        end
    end

    // Read operation
    always @(posedge clk) begin
        if (rst) begin
            rd_ptr <= 0;
            dout   <= 0;
        end else if (rd_en && !empty) begin
```

```
            dout <= fifo_mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
        end
    end

    // FIFO count update
    always @(posedge clk) begin
        if (rst) begin
            fifo_count <= 0;
        end else begin
            case ({wr_en && !full, rd_en && !empty})
                2'b10: fifo_count <= fifo_count + 1; // Write only
                2'b01: fifo_count <= fifo_count - 1; // Read only
                2'b11: fifo_count <= fifo_count;     // Simultaneous read and write
                default: fifo_count <= fifo_count;   // No operation
            endcase
        end
    end

    // Status flags
    assign full         = (fifo_count == FIFO_DEPTH);
    assign empty        = (fifo_count == 0);
    assign almost_full  = (fifo_count >= FIFO_DEPTH - 1);
    assign almost_empty = (fifo_count <= 1);

endmodule
```

```
module alu_top (
    input wire clk,
    input wire rst,
    input wire wr_en,
    input wire [7:0] instr_in,  // 8-bit instruction input to FIFO
    input wire exec_en,          // One-cycle pulse to execute instruction from FIFO
    output [7:0] result
);
    wire [7:0] instr_out;
    wire fifo_empty, fifo_full;

    // Internal control
    wire [2:0] opcode;
    wire [4:0] imm;
    reg [7:0] acc;

    // Instantiate FIFO (Your existing FIFO)
    fifo_sync #(
        .DATA_WIDTH(8),
        .FIFO_DEPTH(16)
    ) fifo_inst (
        .clk(clk),
        .rst(rst),
        .wr_en(wr_en),
        .rd_en(exec_en),
        .din(instr_in),
        .dout(instr_out),
        .full(fifo_full),
        .empty(fifo_empty)
    );

    // Decode instruction
    assign opcode = instr_out[7:5];
    assign imm = instr_out[4:0];

    // ALU logic
    alu_8bit alu_unit (
        .opcode(opcode),
        .acc(acc),
        .imm(imm),
        .result(result)
    );

    // Accumulator update
    always @(posedge clk) begin
        if (rst)
```

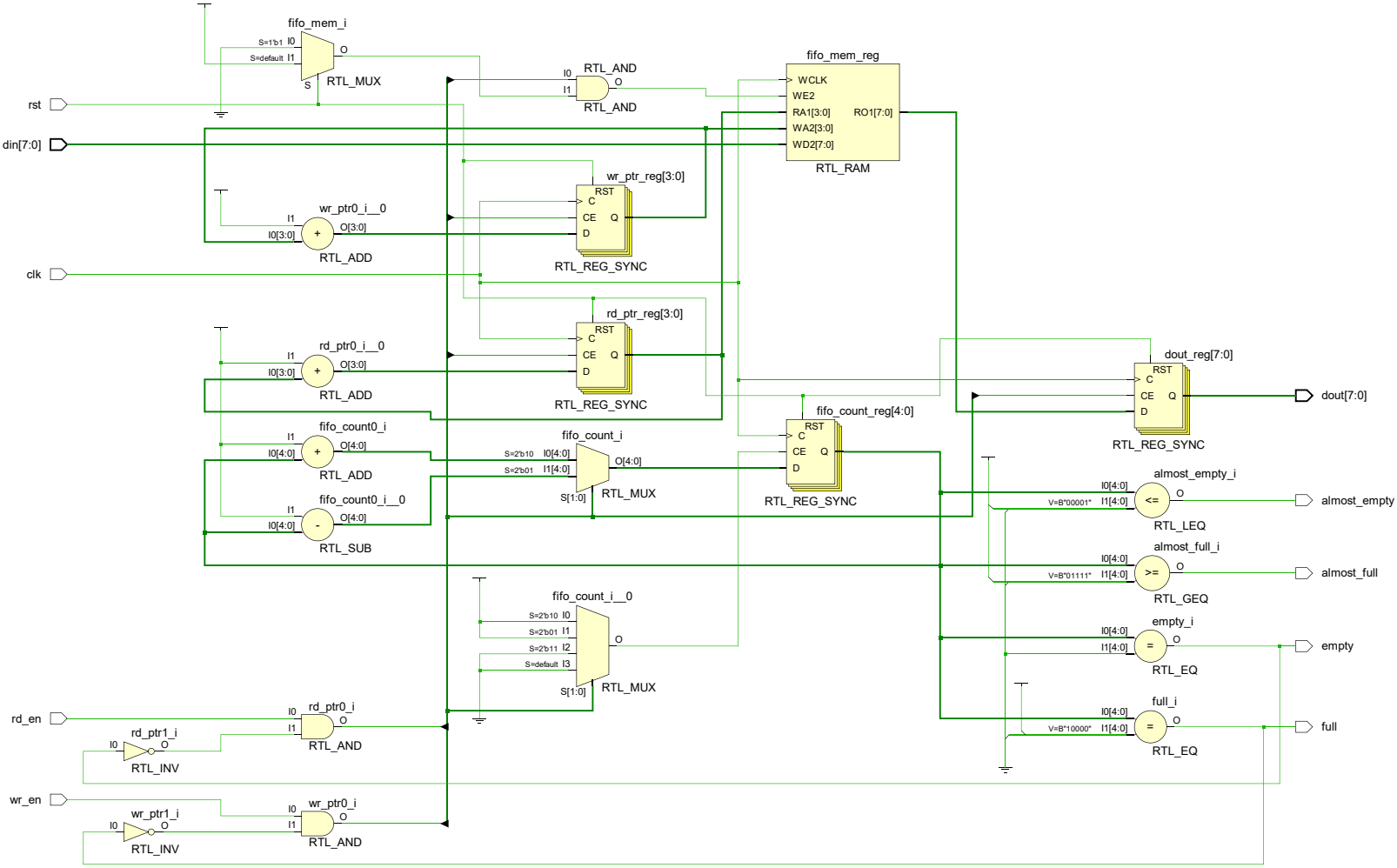
```
            acc <= 0;
        else if (exec_en && !fifo_empty)
            acc <= result;
    end

endmodule
```

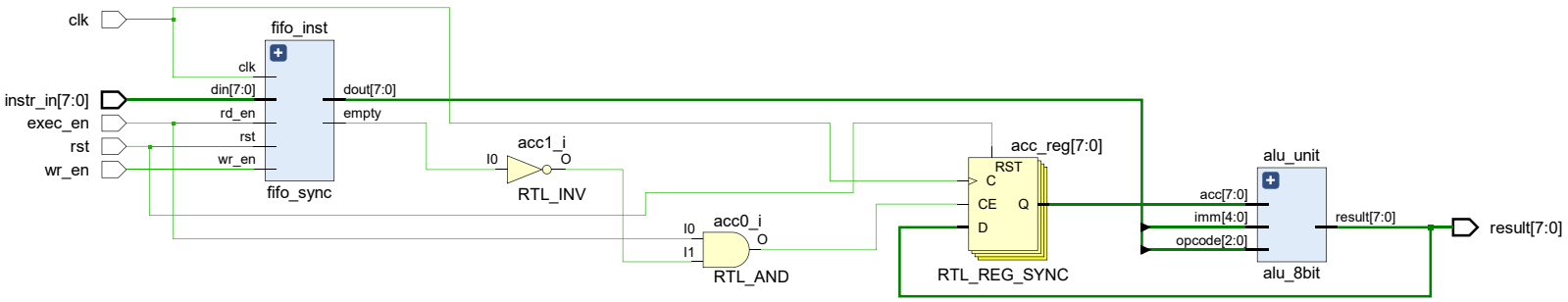
```
module alu_8bit (
    input  wire [2:0] opcode,
    input  wire [7:0] acc,
    input  wire [4:0] imm,       // 5-bit immediate
    output reg  [7:0] result
);
    always @(*) begin
        case (opcode)
            3'b000: result = acc + imm;       // ADDI
            3'b001: result = acc - imm;       // SUBI
            3'b010: result = acc & imm;       // ANDI
            3'b011: result = acc | imm;       // ORI
            3'b100: result = acc ^ imm;       // XORI
            3'b101: result = ~acc;            // NOT (imm ignored)
            3'b110: result = acc + 1;         // INC
            3'b111: result = acc - 1;         // DEC
            default: result = 8'h00;
        endcase
    end
endmodule
```

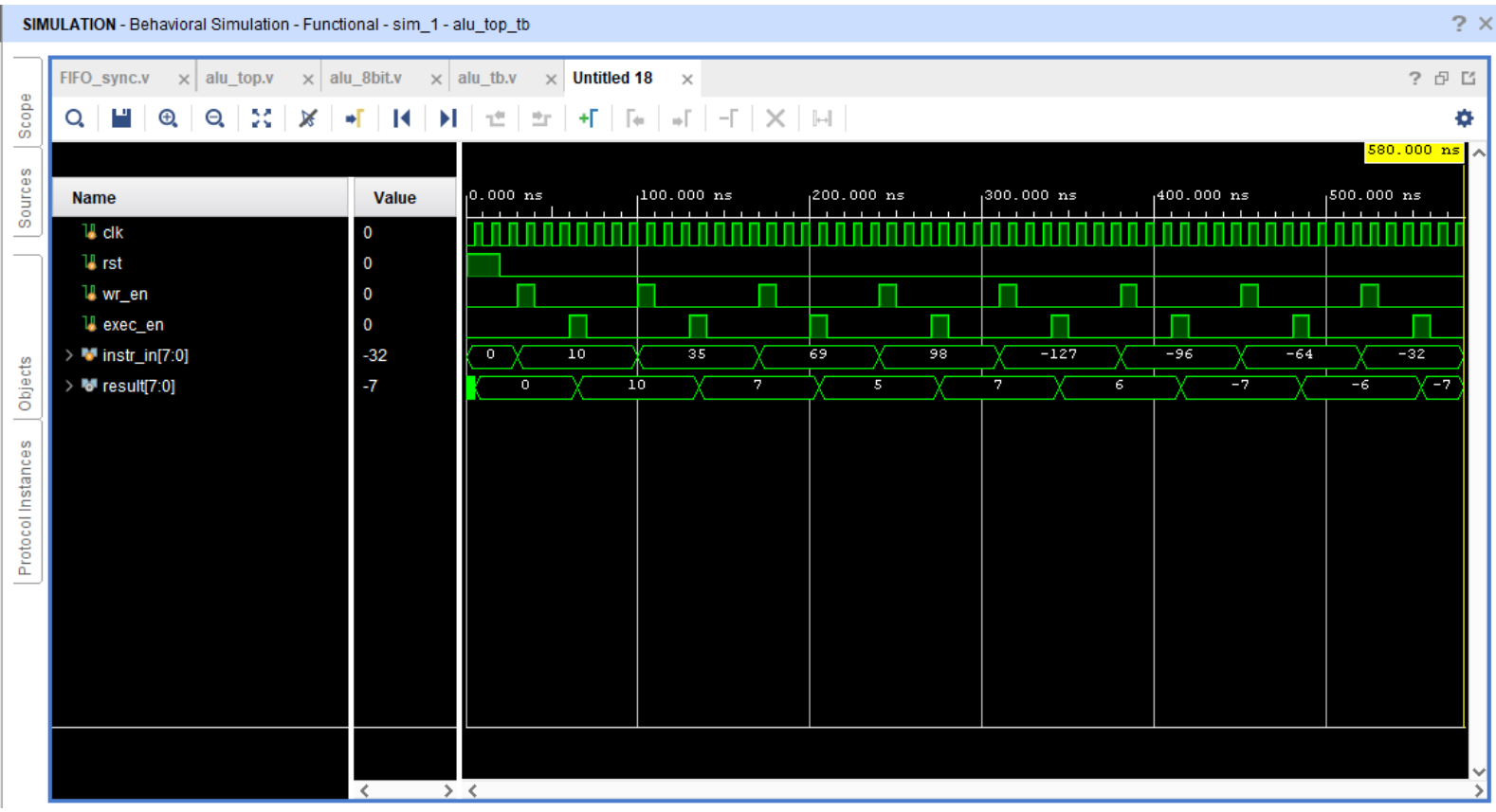
```
`timescale 1ns / 1ps

module alu_top_tb;

    reg clk;
    reg rst;
    reg wr_en;
    reg exec_en;
    reg [7:0] instr_in;
    wire [7:0] result;

    // Instantiate DUT
    alu_top uut (
        .clk(clk),
        .rst(rst),
        .wr_en(wr_en),
        .instr_in(instr_in),
        .exec_en(exec_en),
        .result(result)
    );

    // Clock generation
    always #5 clk = ~clk;  // 100 MHz

    // Instruction encoding helper
    function [7:0] encode_instr;
        input [2:0] opcode;
        input [4:0] imm;
        begin
            encode_instr = {opcode, imm};
        end
    endfunction

    // Apply a single instruction: write to FIFO, then execute
    task apply_instruction;
        input [2:0] opcode;
        input [4:0] imm;
        begin
            @(negedge clk);
            instr_in = encode_instr(opcode, imm);
            wr_en = 1;
            @(negedge clk);
            wr_en = 0;
            repeat(2) @(negedge clk);  // wait until written
            exec_en = 1;
            @(negedge clk);
```

```
        exec_en = 0;
        repeat(2) @(negedge clk);  // wait to observe result
    end
endtask

initial begin
    $display("Starting ALU FIFO Test...");
    $dumpfile("alu_top_tb.vcd");
    $dumpvars(0, alu_top_tb);

    clk = 0;
    rst = 1;
    wr_en = 0;
    exec_en = 0;
    instr_in = 8'h00;

    // Reset the system
    repeat(2) @(negedge clk);
    rst = 0;

    // Test sequence
    // ACC starts at 0

    apply_instruction(3'b000, 5'd10);  // ADDI 10   → ACC = 0 + 10 = 10
    $display("ADDI: Result = %d", result);

    apply_instruction(3'b001, 5'd3);   // SUBI 3    → ACC = 10 - 3 = 7
    $display("SUBI: Result = %d", result);

    apply_instruction(3'b010, 5'd5);   // ANDI 5    → ACC = 7 & 5 = 5
    $display("ANDI: Result = %d", result);

    apply_instruction(3'b011, 5'd2);   // ORI 2     → ACC = 5 | 2 = 7
    $display("ORI : Result = %d", result);

    apply_instruction(3'b100, 5'd1);   // XORI 1    → ACC = 7 ^ 1 = 6
    $display("XORI: Result = %d", result);

    apply_instruction(3'b101, 5'd0);   // NOT       → ACC = ~6 = 8'b11111001 =
249
    $display("NOT : Result = %d", result);

    apply_instruction(3'b110, 5'd0);   // INC       → ACC = 249 + 1 = 250
    $display("INC : Result = %d", result);

    apply_instruction(3'b111, 5'd0);   // DEC       → ACC = 250 - 1 = 249
```

```
        $display("DEC : Result = %d", result);

        $display("ALU FIFO Test completed.");
        $finish;
    end

endmodule
```


fifo_mem_i
S=1'b1 I0
S=default I1
O
S
RTL_MUX
rst
din[7:0]
clk
RTL_AND
I0
I1
O
RTL_AND
fifo_mem_reg
WCLK
WE2
RA1[3:0]
RO1[7:0]
WA2[3:0]
WD2[7:0]
RTL_RAM
wr_ptr_reg[3:0]
RST
C
CE
Q
D
RTL_REG_SYNC
wr_ptr0_i__0
I1
I0[3:0]
O[3:0]
RTL_ADD
rd_ptr_reg[3:0]
RTL_REG_SYNC
rd_ptr0_i__0
RTL_ADD
dout_reg[7:0]
RTL_REG_SYNC
dout[7:0]
fifo_count_reg[4:0]
RTL_REG_SYNC
fifo_count0_i
RTL_ADD
fifo_count_i
S=2'b10 I0[4:0]
S=2'b01 I1[4:0]
O[4:0]
S[1:0]
RTL_MUX
fifo_count0_i__0
RTL_SUB
almost_empty_i
I0[4:0]
V=B"00001" I1[4:0]
<=
RTL_LEQ
almost_empty
almost_full_i
V=B"01111"
>=
RTL_GEQ
almost_full
empty_i
=
RTL_EQ
empty
full_i
V=B"10000"
full
fifo_count_i__0
S=2'b10 I0
S=2'b01 I1
S=2'b11 I2
S=default I3
rd_en
rd_ptr0_i
rd_ptr1_i
RTL_INV
RTL_AND
wr_en
wr_ptr0_i
wr_ptr1_i
RTL_INV
RTL_AND


clk
fifo_inst
clk
instr_in[7:0]
din[7:0]
dout[7:0]
exec_en
rd_en
empty
rst
rst
wr_en
wr_en
fifo_sync
acc1_i
I0
O
RTL_INV
acc0_i
I0
I1
O
RTL_AND
acc_reg[7:0]
RST
C
CE
Q
D
RTL_REG_SYNC
alu_unit
acc[7:0]
imm[4:0]
opcode[2:0]
result[7:0]
alu_8bit
result[7:0]


SIMULATION - Behavioral Simulation - Functional - sim_1 - alu_top_tb
Scope
Sources
Objects
Protocol Instances
FIFO_sync.v
alu_top.v
alu_8bit.v
alu_tb.v
Untitled 18
Name
Value
clk
0
rst
0
wr_en
0
exec_en
0
instr_in[7:0]
-32
result[7:0]
-7
580.000 ns
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
0
10
35
69
98
-127
-96
-64
-32
0
10
7
5
7
6
-7
-6
-7

ALU Operation Calculations

ALU Operation Format

Each instruction is 8 bits:

`[7:5] Opcode  [4:0] Immediate (imm)`

Initial State

$$\text{Accumulator (acc)} = 0$$

Instruction Execution Steps

1. ADDI 10 (Opcode = 000, imm = 10)

$$\text{acc} = 0, \quad \text{imm} = 10 \Rightarrow \text{result} = acc + imm = 0 + 10 = \boxed{10}$$

2. SUBI 3 (Opcode = 001, imm = 3)

$$\text{acc} = 10, \quad \text{imm} = 3 \Rightarrow \text{result} = acc - imm = 10 - 3 = \boxed{7}$$

3. ANDI 5 (Opcode = 010, imm = 5)

$$\text{acc} = 7 = \texttt{00000111}, \quad \text{imm} = 5 = \texttt{00000101}$$

$$\text{result} = acc \,\&\, imm = \texttt{00000101} = \boxed{5}$$

4. ORI 2 (Opcode = 011, imm = 2)

$$\text{acc} = 5 = \texttt{00000101}, \quad \text{imm} = 2 = \texttt{00000010}$$

$$\text{result} = acc \,|\, imm = \texttt{00000111} = \boxed{7}$$

5. XORI 1 (Opcode = 100, imm = 1)

$$\text{acc} = 7 = \texttt{00000111}, \quad \text{imm} = 1 = \texttt{00000001}$$

$$\text{result} = acc \oplus imm = \texttt{00000110} = \boxed{6}$$

6. NOT (Opcode = 101, imm ignored)

$$\text{acc} = 6 = \texttt{00000110}$$

$$\text{result} = \sim acc = \texttt{11111001} = \boxed{249}$$

7. INC (Opcode = 110)

$$\text{acc} = 249 \Rightarrow \text{result} = acc + 1 = \boxed{250}$$

8. DEC (Opcode = 111)

$$\text{acc} = 250 \Rightarrow \text{result} = acc - 1 = \boxed{249}$$

Summary Table of Results

Step	Opcode	Operation	ACC Before	imm	Result	ACC After
1	000	ADDI 10	0	10	10	10
2	001	SUBI 3	10	3	7	7
3	010	ANDI 5	7	5	5	5
4	011	ORI 2	5	2	7	7
5	100	XORI 1	7	1	6	6
6	101	NOT	6	-	249	249
7	110	INC	249	-	250	250
8	111	DEC	250	-	249	249